NUWC-NPT TM 942006



NAVAL UNDERSEA WARFARE CENTER DIVISION
NEWPORT, RI


TECHNICAL MEMORANDUM

# NEW ATTACK SUBMARINE COMBAT SYSTEM DEVELOPMENT PROGRAM: SOFTWARE DEVELOPMENT REUSE TASK

10 February 1994

Prepared by :
Daniel Juttelstad
Technology and Advanced Systems Division
Combat Control System Department

John McGarry
Systems Development Division
Combat Control Systems Department

Steve Roodbeen
Technology and Advanced Systems Division
Combat Control Systems Department

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **10 FEB 1994** | **Technical Memo** | **10-02-1994 to 10-02-1994** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **New Attack Submarine Combat System Development Program : Software Development Reuse Task** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| **Daniel Juttelstad; John McGarry; Steve Roodbeen** | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| **Naval Undersea Warfare Center Division,Newport,RI,02841** | **TM 942006** |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**NUWC2015**

14. ABSTRACT
**This memorandum provides an overview of the software reuse process for the Naval Undersea Warfare Center Division Newport. The three primary areas of the process that are addressed are the domain analysis, reuse software design metrics, and re-engineering software for reuse.**

15. SUBJECT TERMS
**software reuse**

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | |
| **unclassified** | **unclassified** | **unclassified** | **Same as Report (SAR)** | **48** | |

## ABSTRACT

This memorandum provides an overview of the software reuse process for the Naval Undersea Warfare Center Division Newport. The three primary areas of the process that are addressed are the domain analysis, reuse software design metrics, and re-engineering software for reuse.


## ADMINISTRATIVE INFORMATION


This work was performed under Code 22 internal funding.



The authors of this memorandum are located at the Naval Undersea Warfare Center Division, Newport, Rhode Island 02841-1708.



Reverse Blank

## TABLE OF CONTENTS

## LIST OF FIGURES

# 1. INTRODUCTION

## 1.1 REUSE TASKS

The Naval Undersea Warfare Center (NUWC) Division, Newport, Rhode Island, has initiated an applied software development reuse task in support of the New SSN (NSSN) combat system development program. The long term objective of this task is to define, characterize, and manage reusable Ada and other programming language software components in support of the future submarine combat system development process. As currently configured, the software reuse task includes the following activities:

- Functional domain analysis and definition.
- Identification and procurement of available Ada software components that may satisfy domain requirements.
- Characterization of the available software components in terms of reusable software attributes.
- Software re-engineering of the available software components into reusable software assets.
- Cataloging the reusable components into software repositories for easy access by submarine combat system developers.

The reuse task comprises three distinct design and analysis processes that are necessary to realize the objective of reuse . These processes are domain engineering, software design metrics, and re-engineering. Detailed information for each is provided in the chapters that follow. Figure 1 depicts the overall approach to achieving the reuse task objectives. The boxes in the center of the illustration are the functional areas of major concern to be addressed in this document.

Each functional area requires specific personnel expertise. These various expertice areas are identified in the list in Figure 1. They are then mapped to the functional area boxes identifying functional positions.. The number of times a functional position is identified in the figure, does not correlate to the number of personnel positions. One person may be capable of performing multiple functions or working multiple areas. The number pesonnel performing the work is more a function of the size and complexity of the system being addressed.

The areas of expertice necessary to perform the functional positions are:

Software Quality Analyst: Responsible for defining the characterization of the software's quality and applicability to the domain. The software quality analyst also develops the evaluation criteria and acceptance quality level for the candidate software that is to be integrated into the Reuse Repository.

Software Re-Engineer: Responsible for modifying legacy software to increase its quality to the acceptance level for integration into the Reuse RepositoryConcerns of the Software Re-Engineer include dealing with the legacy code existing in CMS-2, and Ada developed utilizing structured development methods and evolving languages such as Ada 9x and C++.

Domain Analyst: Responsible for evaluating candidate systems within the scope of the domain. This process includes the definition of a domain architecture for use by the reuse repository. The domain analyst is also responsible for defining the information requirements for performing domain analysis and working with the system experts in collecting this information.

Repository Librarian: Responsible for developing and maintaining the software repository. This includes mapping the domain architecture and software reuse components into the repository and assisting repository users in retrieving candidate components for target systems.

System Experts: Individuals intimately familiar with the application systems, or portions of systems, within the domain. Responsible for providing necessary information to the domain analyst for developing the generic domain architecture. Responsible for evaluating candidate reusable software for applicability to a target system.

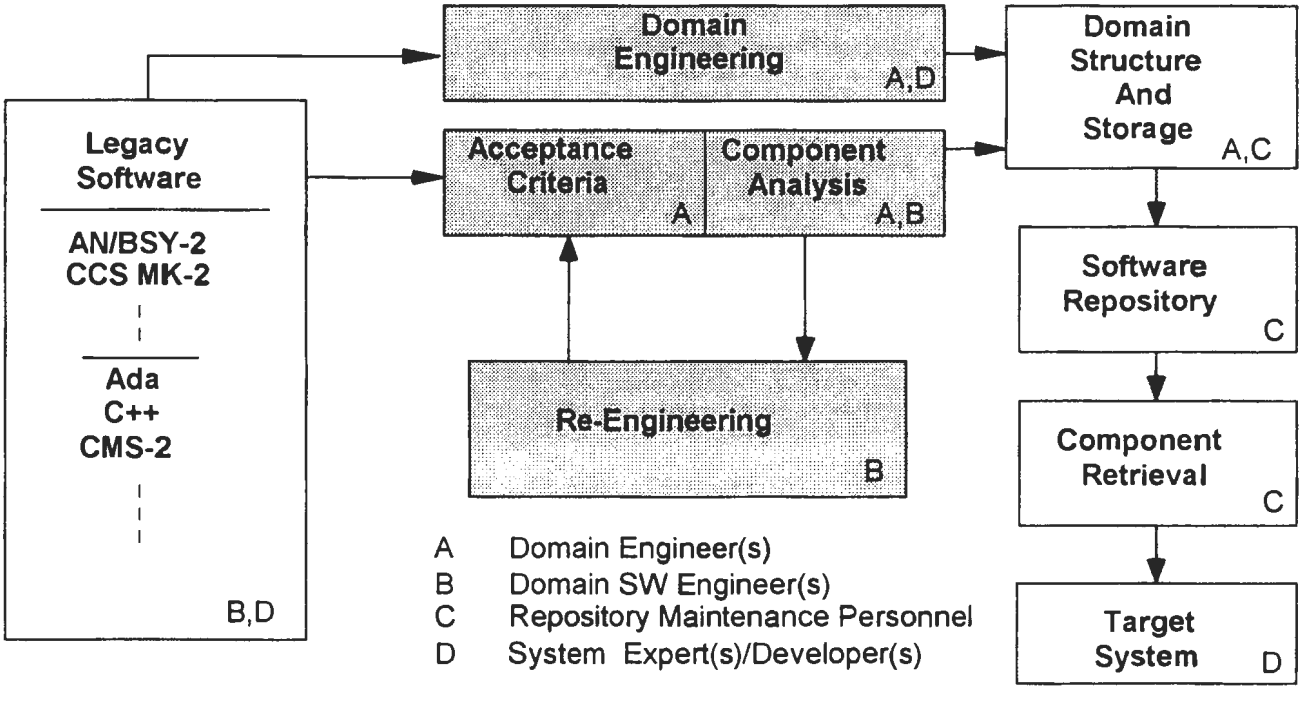


**Figure 1. Reuse Process**

Fiscal Year 1993 tasks focused on the implementation and first level verification of the initial processes and tools required to support the software reuse activities, and technical interchange with parallel system architecture, prototyping, and functional efforts.

## 1.2 DOMAIN MODEL AND DEFINITION

The objective of domain analysis is to develop a generic domain definition. This definition consists of a set of software solutions that apply to the submarine combat system domain. The results will identify the information to be captured from the development of the submarine combat systems software, the software structure, and organization with the purpose of making existing software components reusable. Future systems to be investigated include Combat Control System (CCS) Mk 2 and AN/BSY-1, and AN/BSY-2.

The following functions are inherent in the process of the domain analysis effort:

- Identify information to be captured with respect to the design of the candidate software components development.
- Categorize software components.
- Identify the generic software structure format.

## 1.3 SOFTWARE CHARACTERIZATION

The objective of the software characterization portion of the software reuse task was to develop and validate an overall methodology for defining those characteristics and attributes which support software reuse. This was accomplished by developing and validating specific product and process metrics that correlated to a projected set of software "reuse" attributes.

The following functions are inherent to software characterization:

- Develop software characterization structure, to include a candidate list of "reuse" attributes, associated metrics, and a defined measurement/characterization methodology.
- Identify, procure, and implement automated process and product measurement tools.
- Apply the structure and tools to sample software components.
- Validate the selected reuse attributes and metrics through prototype user feedback.

The initial characterization structure focused on automated product measures. Reusable software component acceptance and re-engineering measures were also addressed.

## 1.4 SOFTWARE RE-ENGINEERING

The objective of the software re-engineering portion of the software reuse task was to develop and validate an overall methodology for the modification of existing Ada software components into reusable components that meet the overall software reuse task requirements. This objective was accomplished by the use and integration of state-of-the-art Ada development tools and environments that support activities such as design recovery, redesign, documentation and Ada code development, integration, and testing.

The software re-engineering task emphasized the following activities:

- Developing a re-engineering model (i.e., methodology).
- Assembleing and integrating a software re-engineering development environment.
- Interpreting characterization data and software system requirement specifications, as they apply to the re-engineering effort.
- Modifying the candidate Ada software components to fit the re-engineering model.
- Submiting the modified (re-engineered) component for re-characterization. If re-engineered components meets or exceed characterization thresholds, enter the component into the repository.
- Evaluate the overall effectiveness of the re-engineering environment and the developed model and document the findings.

## 1.5 REUSE REPOSITORY POPULATION

The initial objective of the repository population effort is to provide an analysis of the candidate software repository and domain analysis tools effectiveness.

The software repository population process maps the generic results of the domain analysis software categories into the candidate repositories. The repository is to be populated by taking the software components that have met the acceptance criteria as defined by the software characterization task. This task was also responsible for obtaining and incorporating the associated information with each software component as defined by the domain analysis and definition task.

# 2. DOMAIN ENGINEERING

## 2.1 DESCRIPTION

Domain engineering differs from system engineering in that it deals with the scope of a common application problem space rather than a specific application development. For the purposes of NUWC, Newport, the domain of interest is the submarine combat system. The legacy systems of interest are the AN/BSY-1, AN/BSY-2 and the CCS Mk 2.

Domain engineering addresses domain analysis, and domain design. The analysis portion addresses evaluating the domain requirements and application designs to identify a common domain model. This domain model represents the domain design. The domainan analysis approach is to capture the existing system designs for use in implementation of a new domain model. The domain design is to identify existing common software components across the submarine combat system domain to be used in the development of the domain model.

## 2.2 DOMAIN ANALYSIS PROCESS

NUWC Newport Code 2221 has developed the approach to collect information about the legacy systems by gathering information from the system experts and reverse engineering the software to extract specific design information and rationale.

### 2.2.1 DOMAIN INQUIRIES

To gather information about the existing legacy systems NUWC Newport Code 2221 has developed a template of information that is to be completed by the component developers, the domain analysts, and the domain engineer for each candidate component. By collecting this data, the domain engineer will have extensive information available for developing the domain model and for eventual inclusions in the repository.

The template for the component design information is provided in appendix A. The format identifies each paragraph for the information and who is responsible for providing or developing the information. For the paragraphs where more than one individual is identified, the order they are presented indicates their level of responsibility with the most responsible listed first. There are three types of individuals identified:

> Component Developer: A person with extensive familiarity with the existing component. This may be the individual responsible for designing and coding the component or maintaining it. These individuals are typically working on the existing systems being evaluated.
>
> Domain Engineer: A person familiar with domain engineering policies and techniques. This individual is responsible for development of the domain model.

Domain Analyst: A person familiar with the characterization of reusable components. This individual is capable of analyzing the component against existing evaluation criteria and analyzing a component's suitability for reuse.

### 2.2.2 REVERSE ENGINEERING

The objective of reverse engineering in domain analysis is to extract legacy system design information. This information is for performing domain analysis and developing the domain model. It is also the first step in identifying candidate components, for re-engineering and reuse. This process of reverse engineering is automated and provides information on software architecture and design from legacy system source code. These diagrams can then be utilized to evaluate software architecture and design of legacy systems and help define commonality across the submarine combat system domain.

For FY93 the Submarine Architecture System Interface Design (SASID) project was used as the prototype for developing and evaluating the reverse engineering capability. The SASID consists of approximately 25K source lines of Ada. Using Object Maker by MARK V Systems as the reverse engineering tool, Buhr 84 Diagrams were developed for the SASID Ada software. The SASID software was in development at the time of the analysis and the diagrams do not represent the SASID design since much of the software shown was stubbed out during the early development. This, however, proved fully useful to the SASID development team as a means to view and evaluate the state of the software at that particular phase of development.

Figure 2 shows examples of the BUHR 84 notation.

Object Maker creates a high level system diagram as depicted in figure 3. This diagram shows the main components of the system. Each icon is bolded since all may be expanded.

Figure 4 presents an expansion of package Geosit. Showing the packages and data that it interfaces with. For example, Geosit Package - Updated_Geo_Model accesses the SPHERE FIDU DATA BASE to obtain the identified data. Again, the bold borders indicate what packages may be expanded for more detail.

SASID is a fairly simple system compared to a deployed combat system. Object Maker requires a closure to obtain complete design information. Therefore, if the systems are large and complex it requires extensive processing resources to generate reverse engineering diagrams as all the software is required to be locally available while running the reverse engineering process.

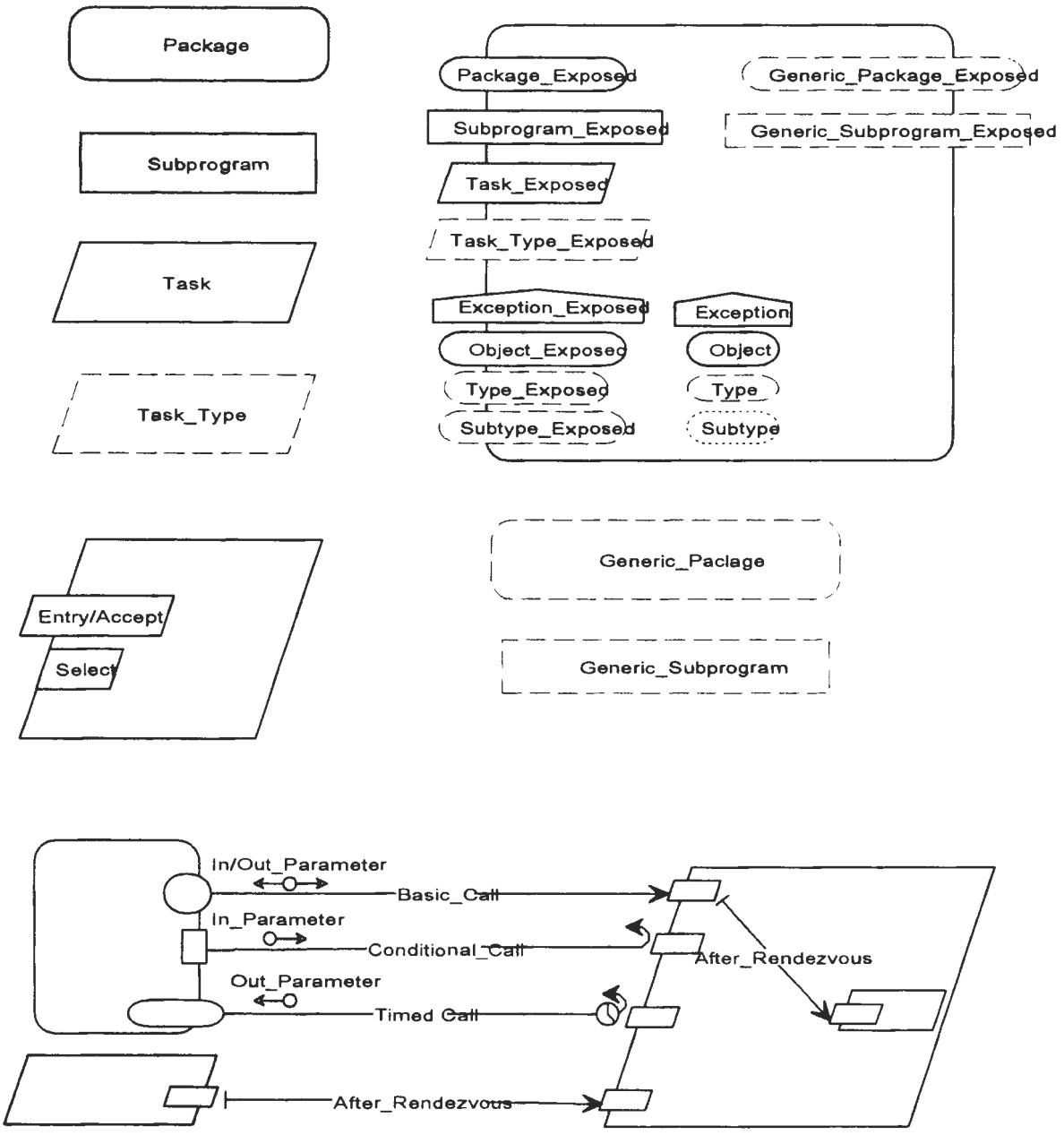


**Figure 2 . Buhr 84 Notation**

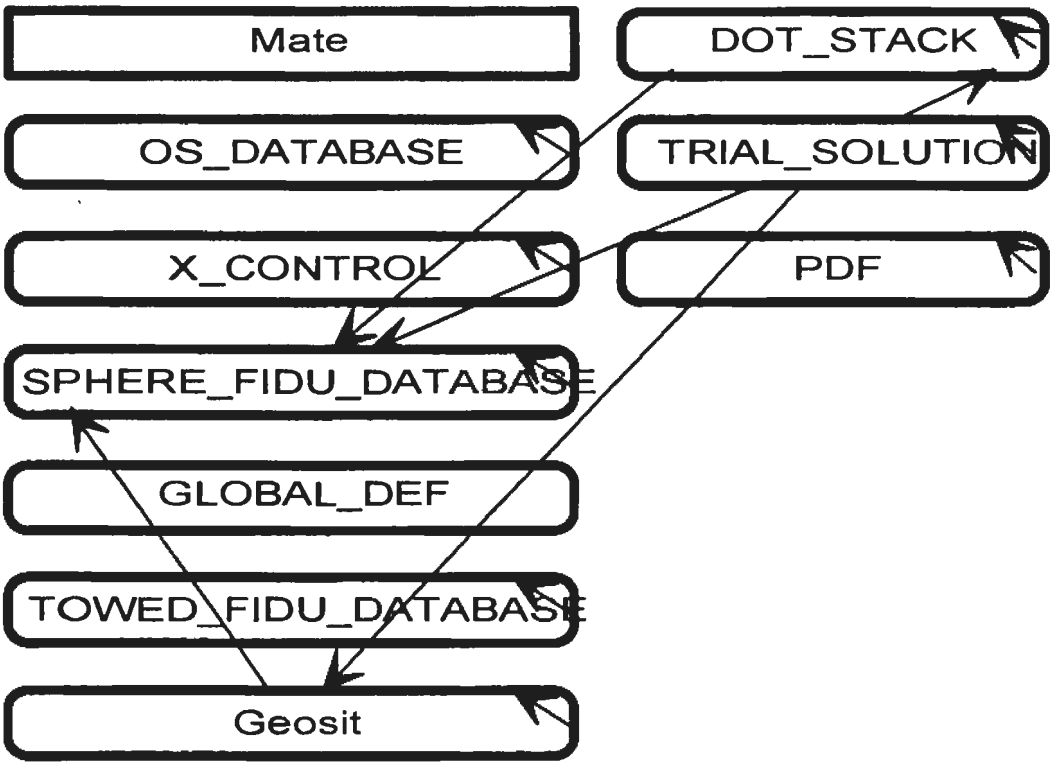


**Figure 3. System Diagram**

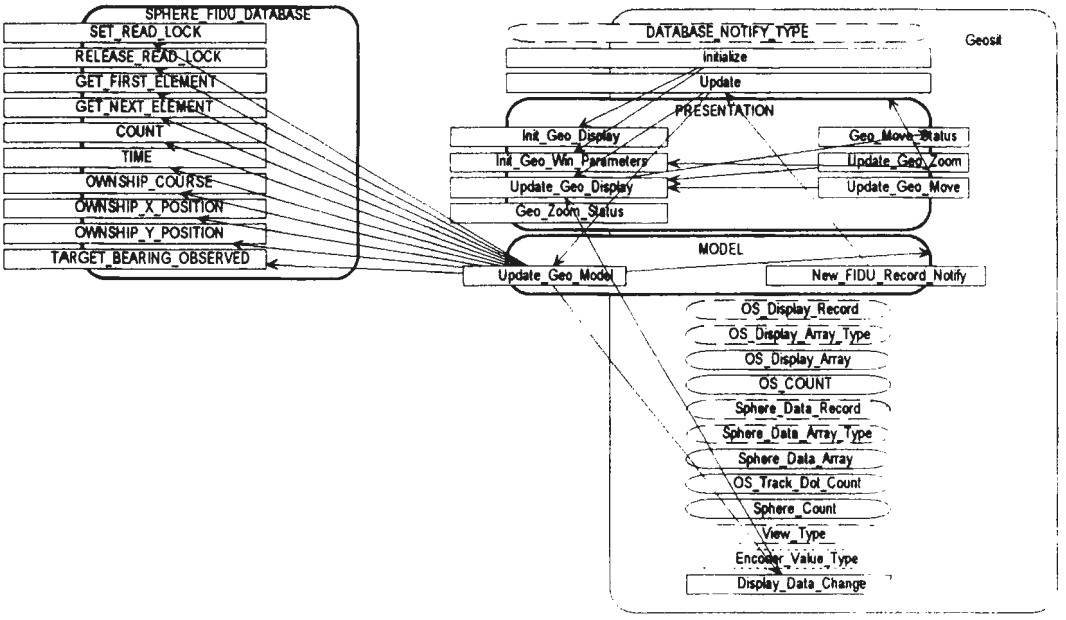


**Figure 4. Package Geosit**

## 2.3 DOMAIN MODEL

The domain model is a logical representation of the system . The model provides system developers of a target system with a view of the domain that allows them to identify and access components within the repository that may be utilized in the target system. For the purposes of the submarine domain model it will be necessary to have multiple representations or views of the model. This requirement provides for queries of the repository for different purposes and different states of the target system's development. The anticipated views for the model are presently a categorization of software functions model, a domain architecture model, and an object categorization model. Presently, work has been done to define a Software Functions Model based on existing domain systems and functions. The present structure of the model is given in appendix B.

The combat systems domain characterization model was developed using AN/BSY-2 design and CCS Mk 2 design for the tactical and Next Generation Computer Resources (NGCR) standards for the system software. This model provides for an Open Systems Architecture (OSA) foundation.

# 3. REUSE INITIATIVE SOFTWARE DESIGN METRICS

## 3.1 DESCRIPTION

The objective of this section is to provide a preliminary list of software process and product metrics pertinent to evaluating the design of reusable software components. This list has been empirically derived from metrics applications experience on past development programs and from projected software reuse characteristics and parameters.

The approach for identifying design for reuse metrics encompasses the following steps:

1. Selection of those software parameter measures that have in the past proven to be useful in evaluating overall software design quality.

2. Mapping the defined measures to those process and product characteristics of software design that appear to support the reuse of software objects.

3. Identification of new, more specific designs for reuse measures.

4. Validation of the identified reuse metrics through application in an actual software reuse program.

The metrics addressed in this report are preliminary in nature and have not yet been validated. Metrics conceptual definitions are emphasized, with specific parameters and measures for each metric to be defined at a later date with respect to actual design processes. No specific reusability model is assumed. As the actual validation environment is defined, the list will be revised based upon the implemented reuse model and actual metrics applications feedback. In general, the metrics relate to software developed using the Ada language, but separate efforts are underway to expand coverage to include C, C++, and CMS-2.

## 3.2 DESIGN METRICS PROCESS

Figure 5 provides an overview of the preliminary list of design for reuse metrics. For discussion purposes, the list is divided into both design process and product metrics. In actuality, the process and product measures have been shown to be highly interrelated. Both are useful in projecting overall software design quality, and both are anticipated to be applicable to specific design for reuse objectives.

Design process metrics encompass those measures that characterize software design activities. These measures are generally presented over time, and reflect the overall volatility, quality and performance of the software design process.

Design product metrics are direct measures of the software design products at any given point in time. These products include design specifications, program design instantiations, and preliminary

source code. The design product metrics focus on quality measures of the reusable software component in terms of both interfaces and internal software design. They also include measures of design traceability, use profile, and computer resource utilization of the reusable software component within the software system.

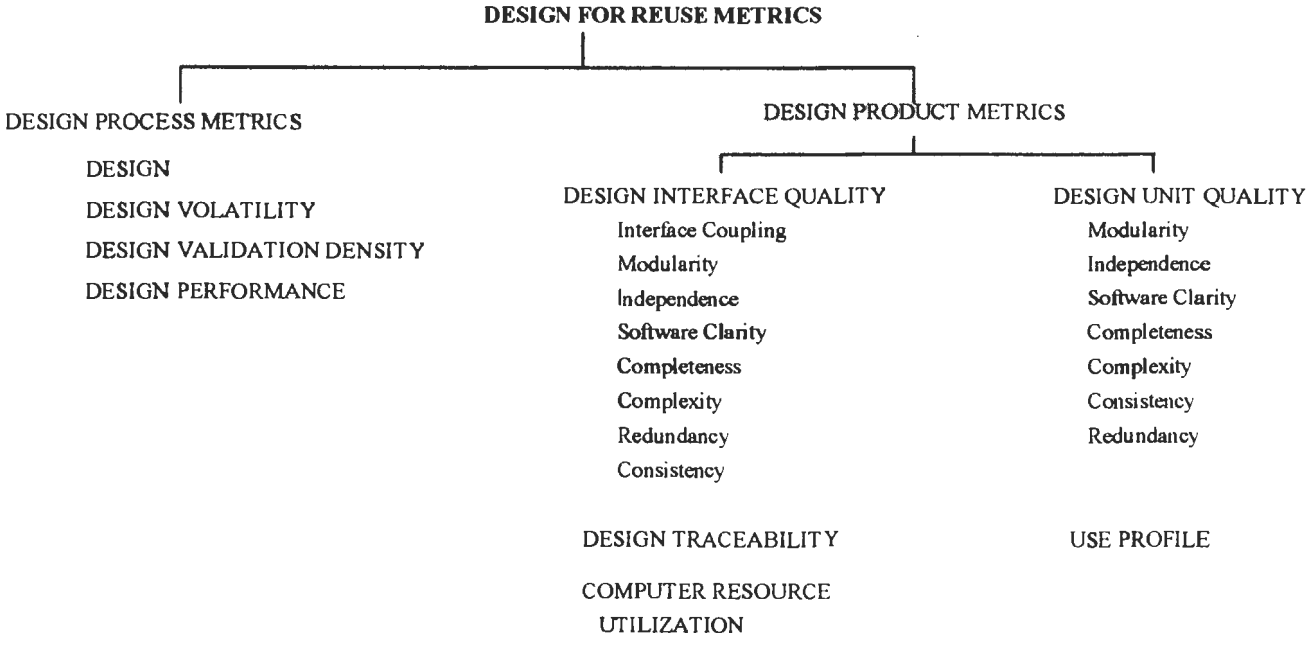


**Figure 5. Design for Reuse Metrics**

The proposed design for reuse process metrics outlined in figure 5 are described as follows:

### 3.2.1 DESIGN DEFECTS

This metric category quantifies the amount and type of defects generated during the software design phase against the design products. It is also applicable during the implementation phase as design changes are backfit into the software design structure. This metric provides insight into the number of defects identified and resolved, into the allocation of those defects by product and development activity, and into the identification of those defects by type (i.e., requirements traceability, standards deviations, design consistency, etc.). The design defect metric essentially provides a general quality profile useful in determining the reusability of the design. For example,

past experience has shown that a high rate of defect discovery within software design products such as design specifications, interface design documents, etc., is indicative of an inadequate or incomplete design structure that materially decreases the design integrity of the software products. Many aspects of code quality are impacted, thereby limiting the software in terms of reuse.

### 3.2.2 DESIGN VOLATILITY

This metric category quantifies the amount and rates of change to the software design products during the design and implementation phases. It also attempts to characterize the source and nature of the changes. Excessive design volatility has shown to be directly related to reductions in software product quality, reliability, and maintainability. Causes of design volatility include highly concurrent development (requirements, design) activities, immature or inexplicit requirements definition, implementation of immature or only top level designs, minimal system and software level design modularity, preliminary performance shortfalls, and excessive domain specific requirements changes. Measures of design volatility address the amount and rates of design change of a defined software component during different software development phases.

### 3.2.3 DESIGN VALIDATION DENSITY

This metric category quantifies the number of specific test cases mapped to software design constructs that are successfully validated. It provides insight into the number of design issues actually verified against the design products or the implemented software component. Design validation density is applicable to all software testing sequences given that a design baseline has been previously defined. The metric is particularly valuable in helping to establish confidence in the design of a reusable software component.

### 3.2.4 DESIGN PERFORMANCE

This metric quantifies the relationship of software design effort, schedule, and product output during the design phase. Although somewhat of an indirect reusability metric, the design performance factor provides insight into the efficiency of the design process for a given software object or software domain source. Experience has shown that design performance metrics are useful indicators in helping to identify those software components more likely to contain design deficiencies and design quality inhibitors. Significantly overlapped requirements and design activities, and premature software implementation, for example, generally result in high design volatility and reduced quality of a software component.

## 3.3 DESIGN PRODUCT METRICS

The proposed design for reuse product metrics outlined in figure 5 is described in the following paragraphs.

### 3.3.1 DESIGN INTERFACE QUALITY

This metric category quantifies the quality characteristics of software design interfaces and the connectivity profile for these interfaces between associated design units. Measurement of software interfaces provides a characterization of reusability with respect to an unmodified software component independent of the underlying implementation. The applicable interfaces are external interfaces that define the services provided by a given software component to its client components. As such, the specific interfaces measured by this metric depend upon the granularity of reuse relative to the nature of the reusable software component.

Reuse granularity ranges from large-scale reuse of systems and subsystems to small-scale reuse of individual program design units. An example of design interface quality in large-scale reuse is a measure of the executive service interfaces provided by a run-time executive. An example of design interface quality in small-scale reuse is a measure of the package specification interface provided by a sorting package.

Design interface quality comprises language specific measures of interface coupling, modularity, independence, software clarity, completeness, complexity, redundancy, and consistency. A general description of each of the metric categories related to measuring design interface quality follows.

1. Interface Coupling. This metric category encompasses characteristics that affect the degree of coupling between the reusable software component and the interfaced software. The degree of coupling is determined by the characteristics of the software interfaces. Interface characteristics that minimize the coupling to other design units facilitate the reuse of the software component. The characteristics of the interfaces are assessed with respect to the means by which a component obtains visibility to each of the following categories of information:

   a. data values (data coupling).
   b. declarations such as types, subprograms, exceptions, etc. (syntactic coupling).
   c. computational behavior (semantic coupling).

2. Modularity. This metric category encompasses the characteristics of information hiding, cohesion, and the size profile of the reusable software component. Information hiding addresses the extent to which implementation details are hidden from the clients of the reusable component. Cohesion addresses the extent to which functional capabilities are partitioned into logical groups of interfaces needed by the same clients. Size profile addresses the extent to which interfaces are provided in manageable size structures.

3. Independence. This metric category encompasses characteristics describing the independence of the reusable software component from the underlying host or target run-time system and

computer architecture. Independence is relevant to measuring reusability in circumstances where the host or target systems may change.

4. Software Clarity. This metric category encompasses characteristics of the reusable software component that provide a clear and understandable description of the program design structure.

5. Completeness. This metric category addresses the presence of interface structures required to support the full spectrum of uses of the interface. For example, profiles of exception declarations provide an indication of whether error detection and handling is supported.

6. Complexity. This metric category encompasses measurements of interface characteristics that simplify the processing required to reuse a software component.

7. Redundancy. This metric category addresses the amount of replication of module, data or control flow structures within a set of design units. Replication reduces the reusability of the software by increasing the difficulty of analyzing the impact of modifications and increasing the effort required to make modifications to multiple locations.

8. Consistency. This metric category addresses the consistent usage of interface structures, both within the interface and in the context of the clients of the interface. Consistency includes measures that indicate the presence of extraneous and error-prone structures in the reusable software component.

### 3.3.2 DESIGN UNIT QUALITY

This metric quantifies the quality characteristics of design units, including module structures, data structures, and control flow structures. The applicable units are the units that provide the implementation for a given reusable software component. Measures of unit quality are applicable to assessments of a given reusable software component in circumstances where modifications may be required in order to reuse the component.

Design unit quality comprises language specific measures of modularity, independence, software clarity, completeness, complexity, consistency, and redundancy. Although similar in concept to the design interface metrics, they are distinguished by their application to units within a given component rather than to external interfaces provided by the component. A general description of each of these seven categories follows. Only those characteristics of the metrics categories that differ from those described in the design interface quality section are addressed.

1. Modularity. This metric category encompasses characteristics of information hiding, cohesion, and the size profile of the reusable software component. Cohesion addresses the organization of module and control flow structures into module structures that are tightly coupled. Size profile addresses the size of units, including both executable and declarative structures.

2. Independence. This metric category encompasses characteristics describing the independence of the reusable software component from the underlying host or target run-time system and computer architecture. Independence of design units includes aspects such as the use of machine code, the use of system dependent library routines, memory management issues, and run-time check suppression.

3. Software Clarity. This metric category encompasses characteristics of the reusable software component that provide a clear and understandable description of the program design structure.

4. Completeness. This metric category addresses the presence of structures indicative of incomplete or extraneous design structures. These include the presence of bodies for subprograms, packages and tasks, and profiles of the occurrence of null statements.

5. Complexity. This metric category encompasses measurements of the complexity of control flow and data flow both within and between units. For example, accessing a data value as both a parameter and as a global value increases the complexity of data flow between units.

6. Consistency. This metric category addresses the consistent use of declarations both within and between units of a reusable software component. Consistency includes measures that indicate the presence of extraneous and error-prone control flow and data flow structures.

7. Redundancy. This metric category addresses the amount of replication of module, data or control flow structures within a set of design units. Replication reduces the reusability of the software by increasing the difficulty of analyzing the impact of modifications and increasing the effort required to make modifications to multiple locations.

### 3.3.3 USE PROFILE

This metric category quantifies the extent of use of a component or structure within a given system. Measures of use are applicable to assessments of the potential usefulness of a given component. They are also applicable to the assessment of the complexity of decoupling a given structure from its uses. Decoupling is performed in the context of Re-Engineering software to improve the interface coupling or to eliminate redundancy.

Use profile is measured as the number of uses of a given component or structure within the system. Profiles of use counts aggregated by various categories are also applicable to specific investigations required in the context of reuse.

### 3.3.4 DESIGN TRACEABILITY

This metric category quantifies the completeness and correctness of the software design and code in terms of the mapping to and from system and software requirements. It is applicable to development methodologies in which requirements are modeled as definable entities, assigned a unique identifier, and mapped to and from the software design and implementation.

Allocation is the mapping of system and software requirements to definable entities in the design. Traceability is the mapping of the definable design entities back to the system and software requirements.

Design traceability is measured as the proportion of entities mapped. The measures are aggregated by various categories as required.

### 3.3.5 COMPUTER RESOURCE UTILIZATION

This metric quantifies the utilization of computer resources to determine the excess capacity available for future changes and enhancements. Applicable measures include quantifications of throughput, proportion utilized of various kinds of processors (CPU, IO, etc.), and the proportion utilized of various kinds of storage. Design unit measures of computer resource utilization are estimates based on dynamic analysis of prototypes and static analysis of design representations.

# 4. RE-ENGINEERING

## 4.1 DESCRIPTION

In the realm of software reuse, re-engineering connotes a variety of meanings, all of which are accurate based on a particular perspective. For example, re-engineering may concentrate on the addition of functionality to a single Ada library unit or attempt to implement a complete system design overhaul. The NUWC Newport software reuse effort is emphasizing re-engineering at the Ada library unit level (quality enhancement) and at the system/subsystem level (process enhancement). As a basis for this effort, NUWC Newport is applying the emerging re-engineering technology to the AN/BSY-1, AN/BQG-5/BSY-2, and the CCS Mk 2.

## 4.2 RE-ENGINEERING PROCESS

Effective software reuse is a process that is based primarily on re-engineering. NUWC Newport's overall objective is the definition and implementation of a software reuse re-engineering specific process. NUWC Newport has defined and implemented the minimal re-engineering process which promotes non-reuse compliant software to reuse compliant software.

### 4.2.1 MEASUREMENT ANALYSIS

Efficient reuse specific re-engineering of legacy systems is dependent on effective measurement. NUWC Newport has developed a measurement and selection process which supports expeditious isolation of reusable legacy software, as well as the identification of re-engineerable software assets. With respect to software reuse, re-engineerable software assets refers to legacy software requiring varying degrees of re-engineering (quality enhancement) to meet the criteria representative of reusable software.

Typically, measurement analysis of legacy software would result from an independent effort. However, successful re-engineering of legacy software is contingent upon a thorough understanding of measurement analysis and the metrics associated with such an analysis. That is, modification of legacy software, to enhance metrics values associated with the software, requires in-depth knowledge of how the metric value is derived from the item being measured.

NUWC Newport has integrated several software analysis tools to enhance the reusable software asset measurement and selection process. These analysis tools include (but are not limited to): AdaMAT, the Rational Environment, and The Software Evaluation Environment for Ada (SEE-Ada).

NUWC Newport has also supported the definition of a software reuse specific set of Ada metrics (associated with the AdaMAT metrics analysis tools). The set of metrics is common to the Department of Defense (DoD). This commonalty is maintained with the aid of the Defense Information Systems Agency (DISA). The current set of software reuse specific Ada metrics is provided in appendix C.

Isolation and selection of reusable and/or re-engineerable software assets from a large software system is arduous at best. However, using the software analysis tools NUWC Newport has assembled, the task is greatly simplified.

Figure 6 is representative of the SEE-Ada's capability to support rapid isolation of reusable and/or re-engineerable software assets. Figure 6 provides a SEE-Ada layers view of the various Ada library units comprising the SASID system. With SEE-Ada it is possible to color-code Ada library units that adhere to certain reusability criteria.

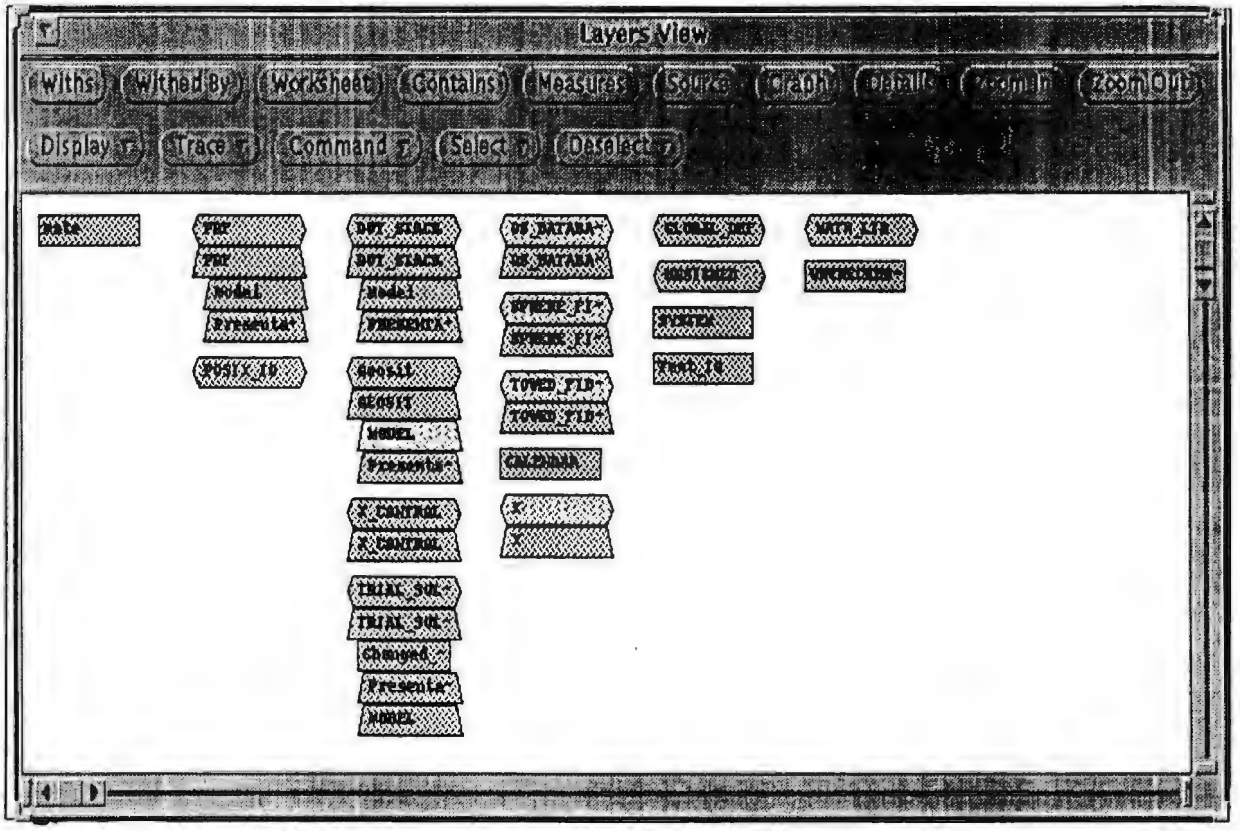


**Figure 6. SEE-Ada: Layers View**

Figure 7 is another view of SASID from SEE-Ada showing the degree to which various Ada library units are coupled. Highly coupled units are probably not potential candidates for reuse. However, the system/subsystem containing such units may be reusable.

Finally, with respect to measurement analysis and metric understanding, the Rational Environment is also useful. The Rational Environment identifies non-reuse compliant software at the source code level, rather than at a modular level like SEE-Ada. However, this Rational Environment capability is ideally suited for supporting re-engineering, as described in the following section.

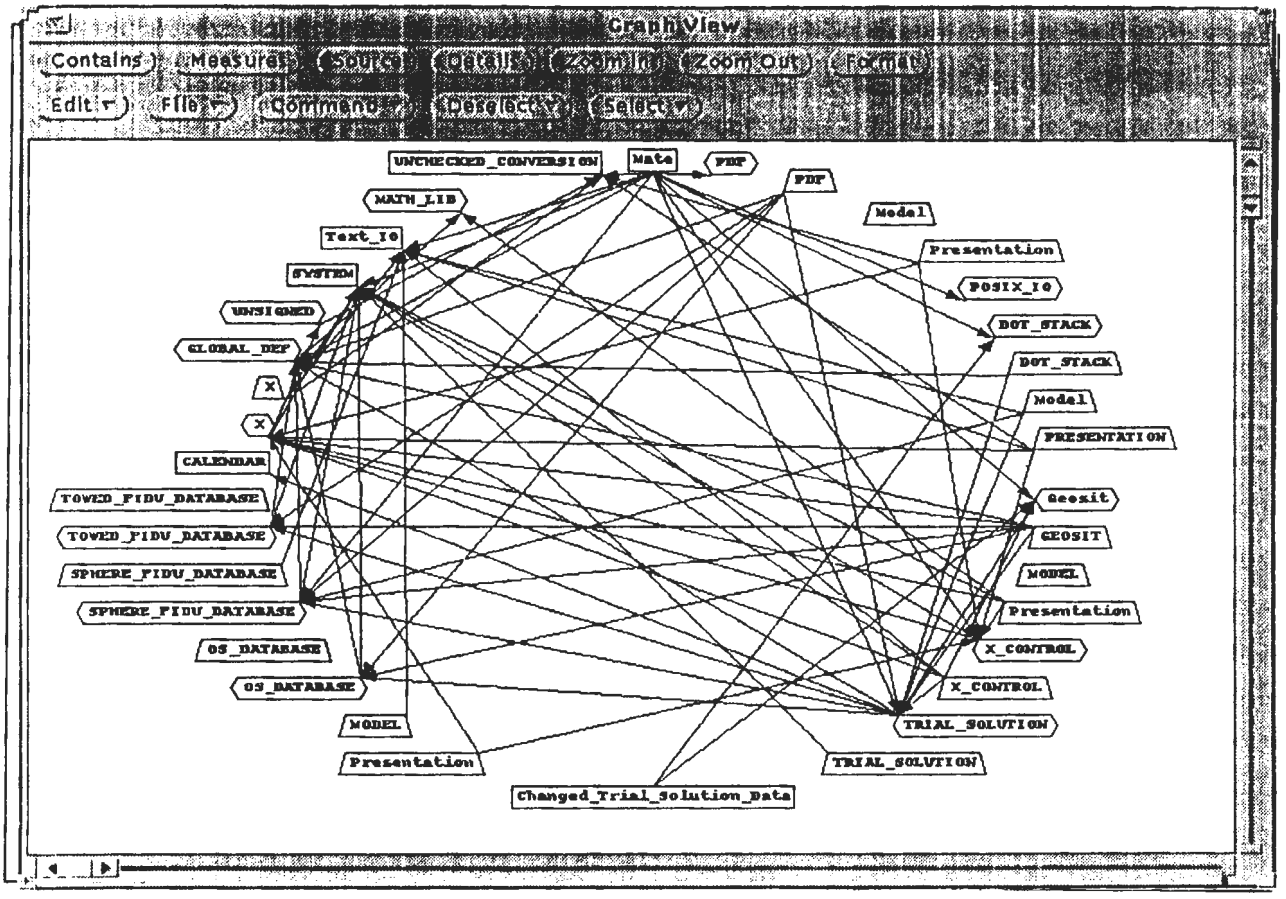


**Figure 7. SEE-Ada: Graph View**

### 4.2.2 RE-ENGINEERING

#### 4.2.2.1 Quality Enhancement

The primary focus of NUWC Newport's software reuse re-engineering effort is the transformation of legacy software into reusable software (i.e., quality enhancement of legacy software with respect to software reuse). The primary software analysis tools supporting this effort are: AdaMAT, the Rational Environment, and SEE-Ada.

Quality enhancement consists of analyzing existing legacy software (i.e., quality measurement with respect to software reuse), modification of legacy software to meet software reuse criteria, and re-measurement to ensure enhancements were effective.

source code:

```
      .
      .
      .
   -- Project:          AN/BSY-2 Submarine Combat System
   --                   Naval Sea Systems Command
   --                   Department of the Navy
      .
      .
      .
   generic

    type DATUM_TYPE is private;

   package LINKED_LIST_UTILITIES is
      .
      .
      .
   private

     type ERROR_TABLE_TYPE;
     type DEF_TABLE_TYPE is access ERROR_TABLE_TYPE;
==>          type Angle is digits 5;  -- No range specified; Impact *
     type ERROR_TABLE_TYPE is
       record
         Left             : Angle;
         Right            : Angle;
         The_Error        : BASE_TYPES.UNSIGNED_INTEGER_16_TYPE;
         The_Error_Rec    : DATUM_TYPE;
         The_Time_First_Error : TIME.TIME_STAMP_TYPE;
       end record;

   end LINKED_LIST_UTILITIES;
```

* **Specifying range and accuracy makes machine dependencies detectable at compile-time.**

This section of AN/BSY-2 Ada source code indicates an inherent non-conformance with established reuse criteria. Namely, no range is given to the type Angle. The following excerpt from an AdaMAT generated characteristics report demonstrates the tools capability to detect such an error.

```
Score   Good   Total |Level------- Metric Name
 0.81    112    138 |1---------- RELIABILITY
 0.78    234    300 |1---------- MAINTAINABILITY
 0.93    608    656 |1---------- PORTABILITY
```

```
0.90    785    868 |1---------- ALL_CRITERIA
  .
  .
  .
0.75     48     64 | 2---------- ANOMALY_MANAGEMENT
0.11      2     18 |  3--------- PREVENTION
  .
  .
  .
0.00      0      1 |  4-------- CONSTRAINED_NUMERICS
                                          -- Score reflects non-conformance
  .
  .
  .
```

The goal of re-engineering is to correct the non-conformance, rendering the Ada library unit more reusable. The following shows the same section of Ada source code after re-engineering:

```
  .
  .
  .
-- Project:          AN/BSY-2 Submarine Combat System
--                   Naval Sea Systems Command
--                   Department of the Navy
  .
  .
  .
-- ACTIVATION SEQUENCE -
generic

  type DATUM_TYPE is private;

package LINKED_LIST_UTILITIES is
  .
  .
  .
private

  type ERROR_TABLE_TYPE;
  type DEF_TABLE_TYPE is access ERROR_TABLE_TYPE;
==>       type Angle is digits 5 range 0.0..360.0;  -- Range/Accuracy Now Explicit
  type ERROR_TABLE_TYPE is
    record
      Left          : Angle;
      Right         : Angle;
      The_Error       : BASE_TYPES.UNSIGNED_INTEGER_16_TYPE;
```

```
    The_Error_Rec       : DATUM_TYPE;
    The_Time_First_Error : TIME.TIME_STAMP_TYPE;
  end record;

end LINKED_LIST_UTILITIES;
```

The fact that the non-conformity has been adjusted is apparent in the new AdaMAT score given to the Ada library unit:

```
Score   Good   Total |Level------- Metric Name

0.83    114    137 |1---------- RELIABILITY
0.79    235    297 |1---------- MAINTAINABILITY
0.93    606    650 |1---------- PORTABILITY
0.91    784    860 |1---------- ALL_CRITERIA
  .
  .
  .
0.79     48     61 | 2---------- ANOMALY_MANAGEMENT
0.19      3     16 |  3--------- PREVENTION
  .
  .
  .
1.00      1      1 |  4-------- CONSTRAINED_NUMERICS
                                          -- Score reflects conformance
  .
  .
  .
```

4.2.2.2 Process Enhancement

NUWC Newport's software reuse re-engineering process enhancement efforts have centered on the acquisition, evaluation, and implementation of the most advanced software analysis tools available. The goal of this effort is to establish an automated software analysis, engineering, re-engineering, and reuse facility. This automated software reuse facility will support the generation and maintenance of new and existing software systems. To date this facility is approximately 50 percent operational.

Experiences with the existing facility have already provided various insights regarding software engineering process improvement and enhancement. As an example, based on various analysis results it is apparent that the extraction of design information from legacy systems is more pertinent to effective software reuse than quality enhancement (as previously described). This observation does not preclude the importance of quality enhancement. Instead, it places a precedence on design extraction. For large scale reuse to be effective, subsystems and systems, as

well as Ada library units, must be reusable. Large scale reuse requires that the design of legacy systems be qualified before the low-level implementation. Specifically, source code quality enhancement should start only after design qualification is complete. This approach provides the most efficient means of extracting the largest possible reusable "pieces" from any given legacy system.

The next level of effort will emphasize automating and improving the process of extracting design information from legacy systems.

## 4.3 NEXT GENERATION RE-ENGINEERING

Analysis of legacy software systems from both a reuse and maintenance perspective has led to the conclusion that the existing functional software engineering paradigm is inadequate. Functional analysis and design techniques are ill-equipped to handle the complexity associated with existing and envisioned Navy software systems. The private sector has reached a similar conclusion and has embraced object-oriented technology (OOT) as the most plausible solution to the complexity management issue. As a result, OOT and its application to Navy software systems is actively being researched. The research is emphasizing both the re-engineering of functionally designed code as object-oriented code and the level of effort required to shift from functionally designed and engineered code to object-oriented designed and engineered code.

With this in mind, the software reuse facility is actively being enhanced to support the OOT paradigm. The facility includes the means to analyze both Ada and non-Ada software in support of the OOT paradigm.

# 5. CONCLUSIONS / ISSUES

## 5.1 ACCESS TO SYSTEM EXPERTISE

Future work must be done to develop a submarine combat system domain architecture and object oriented domain models. The submarine combat system architecture is necessary for defining the domain software architecture, which in turn establishes the definition of the software components for reuse. The software definition philosophy must evolve from the present functional structured approach to an object-oriented . Functional structured definitions have a tendency to require common data structures which are shared by multiple software components. This makes it difficult to extract components from the existing system due to interdependencies of modules with respect to data structures. Object-oriented techniques, on the other hand, encapsulate the data and functionality within the object or software component. This approach facilitates the extraction of the component for reuse purposes.

Component developers' assistance must be obtained to capture knowledge of the existing systems' components. This support will require funding to meet the needs of an evolving model from functional to object-oriented. In order to establish this support, reuse must become a domain issue with domain level support. Individual system development must become a thing of the past and the procurement philosophy must ensure that each new build is done with the intent of providing legacy code for future developments. System developers must remove their narrow view blinders and make their design decisions based on entire domain issues. For this to happen, both policy support and financial support must be provided by the highest levels of management within the submarine combat system domain.

Reverse engineering must be performed on existing combat system software. This requires an evaluation of existing re-engineering resources and existing systems software to determine if the resources must be increased to meet the needs for performing domain analysis. Again, this requires the explicit support of the individual system developers in order to derive the underlying knowledge associated with the software design and development.

## 5.2 INADEQUACY OF EXISTING PROCESSING RESOURCES

Case tools are being developed for performing software engineering, re-engineering, and evaluation. These tools cover the entire software development life-cycle. However, they are designed for use on workstation grade equipment. Presently, there is an inadequate number of desktop processors with capability to access the workstation environments via networking. The NUWC NEWPORT facilities must be expanded to provide the software engineering capabilities to the software engineers desk top in order to effectively incorporate the software discipline into the NUWC software development process.

In addition, Case tools are becoming readily available for Ada, C, and C++, and it is anticipated the market place will develop Ada 9x capability also. However, CMS-2 is a Navy standard

language developed for Navy standard computers. It is doubtful that there will be adequate automated support necessary for the re-engineering of the CMS-2 legacy code.

The metrics concepts presented in this report are based upon prior experience in the measurement of software design processes and products. They form a basis for the development of a software design for reuse metrics set.

New metrics must be developed to address more than software implementation. Since software reuse addresses knowledge capture in all phases of the software development from requirements to implementation, metrics should be developed for determining the reusability of software requirements, specification, and design, in addition to implementation. In order to realize this concept, formal means for stating requirements, specification, and design with a formal language in these areas with well established rules and syntax metrics can be defined that can be used to evaluate the software's quality, completeness, etc.

APPENDIX A
COMPONENT DESIGN SPECIFICATION TEMPLATE

**Component Name: [text]** ***(Component Developer)***

This name should reflect the use of data standardization. If an existing standardized name is in use, it should be used here.

**Required/Optional: [text]** ***(Component Developer)***

This activity identifies components that meet, at least partially, the requirements of the domain model and the imposed constraints.

**Description: [text]** ***(Component Developer)***

The goal of this process is to ascertain the composition of the component systems, and then to identify and describe the components that are common across systems in the domain. The components identified in this process encapsulate services and related information into a single construct with an interface that defines the operations supported by the component.

**Source(s): [text systems/prototypes employing this design]** ***(Component Developer)***

This activity identifies the System that the component was initially developed for.

**Adaptation Requirements (Variants): [text e.g., generic_parameters]** ***(Component Developer)***

The goal of this process is to identify the differences among common components; i.e., adaptation analysis. Adaptation analysis is critical in deriving a domain specific software architecture and component library that can adapt to future system needs. This analysis of required adaptation may be based upon mission, threat, domain, or system planning information.

**Reuse Guidance: [text]** ***(Domain Analyst, Domain Engineer)***

The goal of this process is to determine the reusability of the components by comparison to reusability criteria.

**Lessons Learned: [text]** ***(Component Developer, Domain Engineer)***

The goal for this process is create a set of guidelines for using the domain specific software architecture in a full-scale software development activity. These guidelines include a discussion of the rationale for the selection of a particular alternative, when to use particular components, how/where these components have been used previously, and any lessons learned.

**Constraints:** ***(Component Developer, Domain Engineer)***

Directives/Standards: [text]

The goal of this process is to identify all established and potential constraints affecting the design process. A successful component design enables reuse by meeting as many constraints as economically possible.

This process identifies and records all of the constraints imposed upon the component design activity. A design may be driven by standards, specific software, hardware, tools, directives, variations not defined in the problem space, performance goals, reuse goals, and others. The full set of constraints must be established before the design activity can commence. A successful component design enables reuse by meeting as many constraints as practicable.

Software : [text from SW/HW constraints]
Hardware: [text from sw/hw constraints]
Memory Size Allocation: [text]

**Concurrence: [components_name(s)]** ***(Component Developer)***

The goal of this process is to document component behavior that is characterized with attributes and operations. It also establishes appropriate relationship structures and connections with other components, and portrays and documents required component behaviors and constraints.

**Structure:*****(Domain Engineer, Component Developer)***

The goal of this process is to determine component structures that are common across the domain. System designs and documentation are analyzed and existing system source code is reverse engineered to identify existing structures. Component structure is essential for developing reusable domain models for future system developments.

In this process there are two types of structures that are identified by the analysts: class structures and assembly part structures. "Class structures" imply generalization and specialization (gen-spec) relationships, whereas "assembly structures" imply whole and assembly part (whole-part) relationships. The whole-part diagrams define the composition of the domain, while the class or gen-spec diagrams describe the variation in objects, attributes and services within the domain.

There are several methods and techniques to determine component composition and structure. Traditional functional methods are based on the system functions or functional abstractions. Object-oriented development bases modular software system decomposition on the

classes and components that the system manipulates. Problems arise when attempting to translate from one methodology to another or to mix methodologies.

Whole:[aggregate_component_names] -- if this component has parts
Part-Of: [component_name] -- If this component is part of a larger component
Generalization-of: [component_name]
Specialization-of: [class_name]

**Connection:** ***(Component Developer, Domain Engineer)***

The goal of this process is to determine the necessary connections and interfaces between components in the domain. Connection, essential for determining relationships and dependencies between components, is used to construct reusable component models.

**Instance:** [component_name with cardinality]
**Message:** [component_name with associated service]

**External Interfaces: [component_name with associated attribute]** ***(Domain Engineer, Component Developer)***

The goal of this process is to consolidate and finalize the domain common components model. In addition, rationale and tradeoffs, classification terms, and any pertinent component characteristics (e.g., concurrency, external interfaces) are defined and documented to complete the domain common component model diagrams and specifications.

**State Space: [state transition diagram/matrix]** ***(Component Developer)***

Show the dynamic behavior associated with the component through the use of state transition diagrams and matrix tables. These diagrams show the state space of the component, the events that cause a transition from one state to another, and the actions that result from a state change.

**Attributes:** ***(Component Developer)***

The goal of this process is to ascertain the composition of the domain systems, and then to identify and describe the components that are common across systems in the domain. The components identified in this process encapsulate services and related information into a single construct with an interface that defines the operations supported by the components.

Each component is described in terms of its associated characteristics and behavior. Component characteristics are described in terms of attributes, which define persistent and non-persistent data that an entity manages over time, and state information. Component behavior is described in terms of services or operations that are performed by the component or on the component. Three types of services or operations are identified and investigated by the analyst: constructors, selectors, and iterators. Constructors are operations that alter the state of the

component; selectors are operations that evaluate the current state of the component; and iterators are operations that permit all parts of the component to be traversed.

**Traceability: (Software Architecture)** ***(Domain Engineer)***

This process utilizes components identified in the previous phase to construct the domain specific software architecture. Often several alternatives may be available that satisfy the requirements and constraints. Select the domain specific software architecture that provides the best economic advantage. In very large domains, one overall domain specific software architecture may not provide enough commonality to be of substantial use.

The objective of this activity is to produce a high-level domain specific software architecture depicting the main modules and their interfaces. Domain specific software architectures provide the framework with which to develop tailorable, reusable assets.

Down to Detailed Design/Code: [compilation units - e.g., package specifications]
Up to Domain Model: [problem_space_components, derivations]

**Operations:** ***(Domain Engineer)***

The goal of domain design is to construct a design that reflects the solutions to the problems (requirements) of the domain model within the domain constraints. The domain design consists of domain specific software architecture with reuse guidelines and, optionally, a detailed design.

**Rationale: [text]** ***(Domain Analyst, Domain Engineer)***

The qualities of each domain specific software architecture are measured against identified criteria. These criteria and their weights are established to identify the optimal solutions to the problem space requirements. The advantages and disadvantages of each domain specific software architecture alternative are recorded from a number of trade-off analyses.

**Tradeoffs: [text]** ***(Domain Engineer)***

The goal of this process (and "Rationale" as described above) is to determine and record the qualities of each potential component to support the selection of one or more components from the alternatives.

# APPENDIX B
# COMBAT SYSTEM DOMAIN SOFTWARE CATEGORIES

B.1 TACTICAL
B.1.1 Combat Control
B.1.1.1 Contact Management
B.1.1.1 Contact File Management
B.1.1.1.2 Auxiliary Data Entry
B.1.1.1.3 Multi-Sensor Correlation
B.1.1.1.4 Target Motion Analysis

B.1.1.2 Combat System Management

B.1.1.2.1 Display Select
B.1.1.2.2 Tactical Situation
B.1.1.2.3 Search
B.1.1.2.4 Tactical Support
B.1.1.2.5 Contact Evaluation
B.1.1.2.6 Class Summary

B.1.2 Acoustic
B.1.2.1 Detection
B.1.2.2 Classification
B.1.2.3 Tracking
B.1.2.4 Correlation and TMA
B.1.2.5 Acoustic Support

B.1.3 Weapons
B.1.3.1 Weapons Launched Display and Control
B.1.3.2 Weapons Launched Management

B.1.4 Non-Acoustics
B.1.4.1 External Target Source Communication
B.1.4.2 External Target Source Display Processing
B.1.4.3 External Target Source Functional Processing
B.1.4.4 Ships Own Data Link Processing
B.1.4.5 Ships Tactical Information Data Links Processing

B.1.5 Training

B.2 SYSTEM SERVICES

B.2.1 Capability and Security Services

B.2.2 Data Interchange Services
B.2.3 Event and Error Management Services
B.2.4 File Services
B.2.5 Generalized Input/Output Services
B.2.6 Networks and Communications
B.2.7 Process Management Services
B.2.8 Reliability, Adaptability, and Maintainability Services
B.2.9 Resource Management Services
B.2.10 Synchronization and Scheduling Services
B.2.11 System Initialization and Reinitialization Services
B.2.12 Time Services
B.2.13 Ada language Support Services

B.3 GRAPHICS

B.3.1 Association Table
B.3.2 Bitmap
B.3.3 Color
B.3.4 Colormap
B.3.5 Connection
B.3.6 Context
B.3.7 Cursor
B.3.8 Cut Buffer
B.3.9 Device-Independent Color
B.3.10 Display Macro
B.3.11 Drawing
B.3.12 Error
B.3.13 Event
B.3.14 Extension
B.3.15 Fonts
B.3.16 GC
B.3.17 Host Access
B.3.18 Housekeeping
B.3.19 Image
B.3.20 Image Macro
B.3.21 Internationalization
B.3.22 Keyboard
B.3.23 Keysym Macro
B.3.24 Pixmap
B.3.25 Pointer
B.3.26 Preference
B.3.27 Property
B.3.28 Region
B.3.29 Screen Saver
B.3.30 Selection

B.3.31 Standard Geometry
B.3.32 Text
B.3.33 Visual
B.3.34 Window Location
B.3.35 Window Manager
B.3.36 Display Reource Management

## B.4 DATA BASE REQUIREMENTS

B.4.1 Basic DB Management Services
B.4.1.1 Persistent Data
B.4.1.2 Multiple Users
B.4.1.3 Conventional Alphanumeric Data Types
B.4.1.4 Definition and Manipulation of Binary Large Objects
B.4.1.5 Power of Data Manipulation Language
B.4.1.6 Planned Queries
B.4.1.7 Ad hoc Queries
B.4.1.8 Interactive Queries
B.4.1.9 Embedded Queries
B.4.1.10 Transactions
B.4.1.11 Data Models
B.4.1.12 Conceptual Schema Definition
B.4.1.13 External Schema Definition
B.4.1.14 Mapping to Internal Schema and Database
B.4.1.15 Access Control
B.4.1.16 Heterogeneous Platforms
B.4.1.17 Multiple DBMSs
B.4.1.18 Training Mode
B.4.1.19 Statistical Monitoring

B.4.2 Distribution
B.4.2.1 Distributed Query Processing
B.4.2.2 Distributed Transaction Management
B.4.2.3 Location Transparency
B.4.2.4 Fragmentation Transparency
B.4.2.5 Replication Transparency
B.4.2.6 Data Definition
B.4.2.7 Local Autonomous Processing Capability
B.4.2.8 Continuous Operation
B.4.2.9 Hardware Independence
B.4.2.10 Operating System Independence
B.4.2.11 Network Independence

B.4.3 Heterogeneity
B.4.3.1 Remote Database Access

B.4.3.2 Global Transactions
B.4.3.3 Multi Database Systems
B.4.3.4 Federated Database Systems

B.4.4 Fault Tolerance
B.4.4.1 Retrieval of Fault Information from the DBMS as a DBMS Query
B.4.4.2 Diagnostic Tests
B.4.4.3 Access to the Operational Status of DBMS Components
B.4.4.4 Actions to be Taken on the Occurrence of a Fault

B.4.5 Security
B.4.5.1 Multilevel Security
B.4.5.2 Labeling
B.4.5.3 Mandatory Access Control
B.4.5.4 Discretionary Access Control
B.4.5.5 User Role-Based Access Control
B.4.5.6 Integrity
B.4.5.7 Consistency
B.4.5.8 Identification and Authentication
B.4.5.9 Security Auditing
B.4.5.10 Least Privilege
B.4.5.11 Trusted Path
B.4.5.12 Trusted Recovery
B.4.5.13 Inference and Aggregation
B.4.5.14 Multilevel Data Model
B.4.5.15 SQL Extensions
B.4.5.16 OS Interface
B.4.5.17 Network Interface
B.4.5.18 Heterogeneity
B.4.5.19 Trusted Database Interpretation

B.4.6 Advanced Database Management Services
B.4.6.1 Object Identifiers
B.4.6.2 Binary Large Objects
B.4.6.3 Collection Data type Constructors
B.4.6.4 User-Defined Data Types
B.4.6.5 Sorting Order
B.4.6.6 Temporal Data
B.4.6.7 Spatial Data
B.4.6.8 Uncertain Data
B.4.6.9 Derived Attributes
B.4.6.10 Composite Objects
B.4.6.11 Object Type Hierarchies
B.4.6.12 Object Encapsulation
B.4.6.13 Versions and Configurations

B.4.6.14 Archival Storage
B.4.6.15 Schema Changes
B.4.6.16 Long Transactions
B.4.6.17 Rule Processing
B.4.6.18 Domain-Specific Standards (Enhanced Portability and Interoperability)

## APPENDIX C
## SOFTWARE REUSE SPECIFIC SET OF ADA METRICS

Field definitions for the following metrics:

1) Metric Name Field -- Name of the metric to be controlled and/or analyzed.

2) Include/Exclude Field -- Controls whether or not a given metric is included in the analysis.

3) Reportable/Unreportable Field -- Controls which metrics are reportable. This field differs from the Include/Exclude Field in that unreported metrics are analyzed, but not reported. Excluded metrics are not analyzed and do not contribute metric counts or scores.

4) Adherence Listing Control Field -- Controls which adherences and non-adherences can be reported.
Adherence is relative to particular coding styles, design parameters, quality factors, etc. This field can contain four values: NIL, NON, ADH, or BOTH. NIL indicates that neither adherences nor non-adherences can be reported. NON indicates non-adherences can be reported. ADH indicates
adherences can be reported. Finally, BOTH indicates both adherences and non-adherences are reported.

5) Score Threshold Field -- This field represents the metric score threshold that determines if a metric is
reported. If a particular metric score exceeds the threshold, then that metric is reported.

6) Bad Occurrence Threshold Field -- This field represents the metric bad occurrence threshold. If the
count associated with a given metric is less than this threshold value, then the metric is reported.

7) Total Occurrence Threshold Field -- This field represents the metric total occurrence threshold. If the
count associated with a given metric is less than this threshold value, then the metric is reported.

8) Metric Weighting Field -- This field specifies the weight associated with a given metric, relative to its
parent. Specifically, the weight factor has no effect on the score or count reported for the weighted
metric. However, the weight factor multiplies the effect the weighted metric has on its parent.

9) Desired Occurrence Threshold Field -- This threshold value determines the number of metric occurrences required before a given metric is compared with respect to other metrics.

For more information see the AdaMAT User's Manual and the AdaMAT Reference Manual.

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| RELIABILITY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| MAINTAINABILITY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| PORTABILITY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| ALL_CRITERIA | include | reportable | nil | 100 | 0 | 0 | 1 | |
| SLOC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PHYSICAL_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PHYSICAL_ADA_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| ADA_UNCOMMENTED_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| ADA_COMMENTED_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| COMMENTED_LINES_WITH_TEXT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| COMMENTED_LINES_BLANK | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PHYSICAL_COMMENT_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| COMMENT_LINES_WITH_TEXT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| COMMENT_LINES_BLANK | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PHYSICAL_BLANK_LINES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| LOGICAL_LINES | include | reportable | nil | 100 | 0 | 0 | 0 | |
| STATEMENTS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| EXECUTABLE_STATEMENTS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| DECLARATIVE_STATEMENTS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CONTEXT_CLAUSES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| WITH_CLAUSES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| USE_CLAUSES | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PRAGMAS | include | reportable | nil | 100 | 0 | 0 | 2 | |
| CYCLOMATIC_COMPLEXITY | include | reportable | nil | 100 | 0 | 0 | 6 | 10 |
| MULTIPLE_COND_CYCLOMATIC_COMPLEXITY | include | reportable | nil | 100 | 0 | 0 | 6 | 10 |
| ANOMALY_MANAGEMENT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PREVENTION | include | reportable | nil | 100 | 0 | 0 | 1 | |
| APPLICATIVE_DECLARATIONS | include | reportable | nil | 100 | 0 | 0 | 0 | |
| APPLICATIVE_DECL_SPECIFICATION | include | reportable | nil | 100 | 0 | 0 | 0 | |
| APPLICATIVE_DECL_BODY | include | reportable | nil | 100 | 0 | 0 | 0 | |
| DEFAULT_INITIALIZATION | include | reportable | nil | 100 | 0 | 0 | 0 | |
| DEFAULT_INIT_SPECIFICATION | include | reportable | nil | 100 | 0 | 0 | 0 | |
| DEFAULT_INIT_BODY | include | reportable | nil | 100 | 0 | 0 | 0 | |
| NORMAL_LOOPS | include | reportable | nil | 100 | 0 | 0 | 2 | |
| CONSTRAINED_SUBTYPE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CONSTRAINED_NUMERICS | include | reportable | nil | 100 | 0 | 0 | 6 | |
| CONSTRAINED_VARIANT_RECORDS | include | reportable | nil | 100 | 0 | 0 | 0 | |
| READ_ONLY_OBJECTS_CONSTANT | include | reportable | non | 100 | 0 | 0 | 1 | |
| READ_ONLY_OBJECTS_IN_SPEC_CONSTANT | include | reportable | non | 100 | 0 | 0 | 1 | |
| READ_ONLY_OBJECTS_IN_BODY_CONSTANT | include | reportable | non | 100 | 0 | 0 | 1 | |
| READ_ONLY_PARAMETERS_IN_MODE | include | reportable | non | 100 | 0 | 0 | 2 | |
| UNREAD_PARAMETERS_OUT_MODE | include | reportable | non | 100 | 0 | 0 | 2 | |
| OUT_PARAMETERS_UPDATED | include | reportable | non | 100 | 0 | 0 | 6 | |
| READ_VARIABLES_DEFINED | include | reportable | non | 100 | 0 | 0 | 1 | |
| READ_VARIABLES_IN_SPEC_DEFINED | include | reportable | non | 100 | 0 | 0 | 6 | |
| READ_VARIABLES_IN_BODY_DEFINED | include | reportable | non | 100 | 0 | 0 | 6 | |
| VARIABLES_READ | include | reportable | non | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_SPEC_READ | include | reportable | non | 100 | 0 | 0 | 2 | |
| VARIABLES_IN_BODY_READ | include | reportable | non | 100 | 0 | 0 | 2 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| DETECTION | include | reportable | nil | 100 | 0 | 0 | 1 | |
| SUPPRESS_PRAGMA | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CONSTRAINT_ERROR | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PROGRAM_ERROR | include | reportable | nil | 100 | 0 | 0 | 6 | |
| STORAGE_ERROR | include | reportable | nil | 100 | 0 | 0 | 6 | |
| NUMERIC_ERROR | include | reportable | nil | 100 | 0 | 0 | 6 | |
| USER_TYPES | include | reportable | nil | 100 | 0 | 0 | 1 | |
| USER_TYPES_FOR_PARAMETERS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| USER_TYPES_SPECIFICATION | include | reportable | nil | 100 | 0 | 0 | 1 | |
| USER_TYPES_BODY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| RECOVERY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| USER_DEFINED_EXCEPTIONS_RAISED | include | reportable | non | 100 | 0 | 0 | 6 | |
| USER_EXCEPTIONS_RAISED | include | reportable | nil | 100 | 0 | 0 | 2 | |
| INDEPENDENCE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| IO_INDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MISSED_CLOSE | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NO_SYS_DEP_IO | include | reportable | nil | 100 | 0 | 0 | 0 | |
| IO_NON_MIX | include | reportable | nil | 100 | 0 | 0 | 0 | |
| TASK_INDEP | include | reportable | nil | 100 | 0 | 0 | 0 | |
| NO_TASK_STMT | include | reportable | nil | 100 | 0 | 0 | 0 | |
| TASK_STMT_NON_MIX | include | reportable | nil | 100 | 0 | 0 | 0 | |
| MACH_INDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| MACHARITHINDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PACKAGE_ARITH_INDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MAX_INT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MIN_INT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MAX_DIGITS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MAX_MANTISSA | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_FINE_DELTA | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_TICK | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_INTEGER_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_SHORT_INTEGER_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_LONG_INTEGER_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_FLOAT_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_SHORT_FLOAT_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_LONG_FLOAT_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_NATURAL_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_POSITIVE_DECL | include | reportable | nil | 100 | 0 | 0 | 1 | |
| FIXED_CLAUSE | include | reportable | nil | 100 | 0 | 0 | 6 | |
| MACHREPINDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_PRAGMA_PACK | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NUMERIC_CONSTANT_DECL | include | reportable | nil | 100 | 0 | 0 | 0 | |
| NUMERIC_TYPE_DECLARATIONS | include | reportable | nil | 100 | 0 | 0 | 0 | |
| CLAUSE_REP_INDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_LENGTH_CLAUSE_FOR_SIZE | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NO_LENGTH_CLAUSE_FOR_STORAGE_SIZE | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NO_ALIGNMENT_CLAUSE_FOR_RECORD_TYPES | include | reportable | nil | 100 | 0 | 0 | 0 | |
| NO_COMPONENT_CLAUSE_FOR_RECORD_TYPES | include | reportable | nil | 100 | 0 | 0 | 2 | |
| MACHCONFIGINDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_ADDRESS_CLAUSE_IN_DECL | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NO_PRAG_SYS_PARAM | include | reportable | nil | 100 | 0 | 0 | 6 | |
| NO_REP_ATTRIBUTE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| MACHCODEINDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| NO_MACH_CODE_STMT | include | reportable | nil | 100 | 0 | 0 | 3 | |
| SOFT_INDEP | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_SYS_DEP_MOD | include | reportable | nil | 100 | 0 | 0 | 0 | |
| NO_IMPL_DEP_PRAGMAS | include | reportable | nil | 100 | 0 | 0 | 6 | |
| NO_PRAGMA_INTERFACE | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NON_ACCESS_TYPE | include | reportable | nil | 100 | 0 | 0 | 2 | |
| NO_IMPL_DEP_ATTRS | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PHYS_LIM_INDEP | include | reportable | nil | 100 | 0 | 0 | 0 | |
| COMPILER_LIMIT | include | reportable | nil | 100 | 0 | 0 | 0 | |
| WITH_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 10 |
| USE_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 10 |
| TRANSFER_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 500 |
| INSTANTIATION_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 10 |
| WHEN_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 100 |
| DECLARATIVE_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 100 |
| EXECUTABLE_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 100 |
| PRAGMA_COMPILER_LIMIT_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 0 | 10 |
| MODULARITY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| INFORMATION_HIDING | include | reportable | nil | 100 | 0 | 0 | 1 | |
| HIDDEN_INFORMATION | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CONSTANTS_HID | include | reportable | nil | 100 | 0 | 0 | 1 | |
| EXCEPTIONS_HID | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_HID | include | reportable | nil | 100 | 0 | 0 | 2 | |
| TYPES_HID | include | reportable | nil | 100 | 0 | 0 | 1 | |
| SUBTYPES_HID | include | reportable | nil | 100 | 0 | 0 | 1 | |
| TASKS_HID | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PRIVATE_INFORMATION | include | reportable | nil | 100 | 0 | 0 | 1 | |
| PRIVATE_TYPES | include | reportable | nil | 100 | 0 | 0 | 0 | |
| LIMITED_PRIVATE_TYPES | include | reportable | nil | 100 | 0 | 0 | 0 | |
| PRIVATE_TYPE_AND_PART | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PRIVATE_TYPE_AND_CONSTANT | include | reportable | nil | 100 | 0 | 0 | 6 | |
| PROFILE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| LIMITED_SIZE_PROFILE | include | reportable | nil | 100 | 0 | 0 | 1 | 100 |
| STATEMENT_PROFILE | include | reportable | nil | 100 | 0 | 0 | 6 | |
| DECLARATION_PROFILE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| LIBRARY_CLAUSE_PROFILE | include | reportable | nil | 100 | 0 | 0 | 6 | |
| SIMPLE_BLOCKS | include | reportable | nil | 100 | 0 | 0 | 6 | |
| COUPLING | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_MULTIPLE_TYPE_DECLARATIONS | include | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NO_VARIABLE_DECLARATIONS_IN_SPEC | include | reportable | nil | 100 | 0 | 0 | 6 | |
| INFORMATION_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| WITHS_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| WITHS_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| WITHS_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| WITHS_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTITIES_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| ENTITIES_IN_SPEC_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| CONSTANTS_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| SUBTYPES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| PROCEDURES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| FUNCTIONS_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| PACKAGES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PROCEDURES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_FUNCTIONS_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_PACKAGES_IN_SPEC_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| ENTITIES_IN_BODY_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| CONSTANTS_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| SUBTYPES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| PROCEDURES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| FUNCTIONS_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| PACKAGES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PROCEDURES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_FUNCTIONS_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_PACKAGES_IN_BODY_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| ENTITIES_IN_SUBUNIT_LOCALIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| CONSTANTS_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| SUBTYPES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| PROCEDURES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| FUNCTIONS_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| PACKAGES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PROCEDURES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_FUNCTIONS_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| GENERIC_PACKAGES_IN_SUBUNIT_LOCALIZED | include | reportable | non | 100 | 0 | 0 | 1 | |
| SELF_DESCRIPTIVENESS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| COMMENTS | include | reportable | nil | 100 | 0 | 0 | 0 | |
| N_COMMENTS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCS_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCS_PACKAGE_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_TASK_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_SUBPROG_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCS_PACKAGE_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_TASK_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_SUBPROG_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_SUBUNIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BODY_STUB | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_STATEMENTS | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCS_EXIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_RETURN | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_GOTO | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_ABORT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_DELAY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_TERMINATE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_WITH | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_USE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_DECLARATIONS | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCS_PRAGMA | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_RECORD_REPRESENTATION | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_ADDRESS_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_ALIGNMENT_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| NCS_LENGTH_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_CONSTANT_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_VARIABLE_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_ENTRY_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 2 |
| NCS_BEFORE_PACKAGE_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BEFORE_TASK_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BEFORE_SUBPROG_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_PACKAGE_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BEFORE_TASK_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BEFORE_SUBPROG_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_SUBUNIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| NCS_BEFORE_BODY_STUB | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_EXIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_RETURN | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_GOTO | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_ABORT | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_DELAY | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_TERMINATE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_WITH | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_USE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_PRAGMA | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_RECORD_REPRESENTATION | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_ADDRESS_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_ALIGNMENT_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_LENGTH_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | 3 |
| NCS_BEFORE_CONSTANT_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_VARIABLE_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 1 |
| NCS_BEFORE_ENTRY_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | 2 |
| N_COMMENTED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_PACKAGE_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_TASK_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_SUBPROG_SPEC | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_PACKAGE_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_TASK_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_SUBPROG_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_SUBUNIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BODY_STUB | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_STATEMENTS | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_EXIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_RETURN | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_GOTO | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_ABORT | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_DELAY | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_TERMINATE | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_WITH | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_USE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_DECLARATIONS | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_PRAGMA | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_RECORD_REPRESENTATION | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_ADDRESS_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_ALIGNMENT_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| NCO_LENGTH_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_CONSTANT_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_VARIABLE_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_ENTRY_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_PACKAGE_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_TASK_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_SUBPROG_SPEC | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_PACKAGE_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_TASK_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_SUBPROG_BODY | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_SUBUNIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_BODY_STUB | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_EXIT | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_RETURN | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_GOTO | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_ABORT | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_BEFORE_DELAY | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_BEFORE_TERMINATE | exclude | reportable | nil | 100 | 0 | 0 | 2 | |
| NCO_BEFORE_WITH | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_USE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_PRAGMA | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_RECORD_REPRESENTATION | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_ADDRESS_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_ALIGNMENT_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_LENGTH_CLAUSE | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_CONSTANT_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_VARIABLE_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| NCO_BEFORE_ENTRY_DECL | exclude | reportable | nil | 100 | 0 | 0 | 1 | |
| IDENTIFIER | include | reportable | nil | 100 | 0 | 0 | 3 | |
| NO_PREDEFINED_WORDS | include | reportable | nil | 100 | 0 | 0 | 3 | |
| SIMPLICITY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CODING_SIMPLICITY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| SIMPLE_BOOLEAN_EXPRESSION | include | reportable | nil | 100 | 0 | 0 | 0 | |
| EXPRES_TO_DO_BOOLEAN_ASSIGN | include | reportable | nil | 100 | 0 | 0 | 1 | |
| DESIGN_SIMPLICITY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| CALLS_TO_PROCEDURES | include | reportable | nil | 100 | 0 | 0 | 1 | 10 |
| ARRAY_TYPE_EXPLICIT | include | reportable | nil | 100 | 0 | 0 | 3 | |
| SUBTYPE_EXPLICIT | include | reportable | nil | 100 | 0 | 0 | 3 | |
| ARRAY_RANGE_TYPE_EXPLICIT | include | reportable | nil | 100 | 0 | 0 | 3 | |
| DECLARATIONS_CONTAIN_LITERALS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| FLOW_SIMPLICITY | include | reportable | nil | 100 | 0 | 0 | 1 | |
| BRANCH_CONSTRUCTS | include | reportable | nil | 100 | 0 | 0 | 1 | 10 |
| SINGLE_EXIT_SUBPROGRAM | include | reportable | nil | 100 | 0 | 0 | 1 | |
| FOR_LOOPS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| LEVEL_OF_NESTING | include | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| LEVEL_OF_NESTING_BY_MODULE | include | reportable | nil | 100 | 0 | 0 | 1 | 5 |
| STRUCTURED_BRANCH_CONSTRUCT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NON_BACK_BRANCH_CONSTRUCT | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_LABELS | include | reportable | nil | 100 | 0 | 0 | 6 | 0 |
| DECISIONS | include | reportable | nil | 100 | 0 | 0 | 1 | 10 |
| GOTOS | include | reportable | nil | 100 | 0 | 0 | 6 | 0 |
| BRANCH_AND_NESTING | include | reportable | nil | 100 | 0 | 0 | 1 | |
| SYSTEM_CLARITY | include | reportable | nil | 100 | 0 | 0 | 1 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| STYLE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| EXPRESSION_STYLE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NON_NEGATED_BOOLEAN_EXPRESSIONS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| EXPRESSIONS_PARENTHESIZED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_WHILE_LOOPS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| FOR_LOOPS_WITH_TYPE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| DECLARATION_STYLE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NO_DEFAULT_MODE_PARAMETERS | include | reportable | nil | 100 | 0 | 0 | 3 | |
| PRIVATE_ACCESS_TYPES | include | reportable | nil | 100 | 0 | 0 | 2 | |
| SINGLE_OBJECT_DECLARATION_LISTS | include | reportable | nil | 100 | 0 | 0 | 3 | |
| SINGLE_IMPLICIT_TYPE_ARRAY | include | reportable | nil | 100 | 0 | 0 | 6 | |
| NO_INITIALIZATION_BY_NEW | include | reportable | nil | 100 | 0 | 0 | 6 | |
| NAMING_STYLE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| STRUCTURES_NAMED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NAMED_LOOPS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NAMED_BLOCKS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| STRUCTURE_ENDS_WITH_NAME | include | reportable | nil | 100 | 0 | 0 | 1 | |
| MODULE_END_WITH_NAME | include | reportable | nil | 100 | 0 | 0 | 3 | |
| LOOP_END_WITH_NAME | include | reportable | nil | 100 | 0 | 0 | 1 | |
| BLOCK_END_WITH_NAME | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NAMED_EXITS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| NAMED_AGGREGATE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| QUALIFICATION_STYLE | include | reportable | nil | 100 | 0 | 0 | 1 | |
| QUALIFIED_AGGREGATE | include | reportable | nil | 100 | 0 | 0 | 3 | |
| QUALIFIED_SUBPROGRAM | include | reportable | nil | 100 | 0 | 0 | 1 | |
| EXACTNESS | include | reportable | nil | 100 | 0 | 0 | 1 | |
| WITHS_UTILIZED | include | reportable | nil | 100 | 0 | 0 | 0 | |
| WITHS_IN_SPEC_UTILIZED | include | reportable | non | 100 | 0 | 0 | 0 | |
| WITHS_IN_BODY_UTILIZED | include | reportable | non | 100 | 0 | 0 | 0 | |
| WITHS_IN_SUBUNIT_UTILIZED | include | reportable | non | 100 | 0 | 0 | 0 | |
| ENTITIES_REFERENCED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| ENTITIES_IN_SPEC_REFERENCED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| CONSTANTS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| SUBTYPES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENUMERATION_LITERALS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| COMPONENTS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| PROCEDURES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| FUNCTIONS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| IN_PARAMETERS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| OUT_PARAMETERS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| IN_OUT_PARAMETERS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| PACKAGES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTRIES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| EXCEPTIONS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| GENERIC_PROCEDURES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_FUNCTIONS_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PACKAGES_IN_SPEC_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTITIES_IN_BODY_REFERENCED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| CONSTANTS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |

| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|
| SUBTYPES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENUMERATION_LITERALS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| COMPONENTS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| PROCEDURES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| FUNCTIONS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| IN_PARAMETERS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| OUT_PARAMETERS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| IN_OUT_PARAMETERS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| PACKAGES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTRIES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| EXCEPTIONS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| GENERIC_PROCEDURES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_FUNCTIONS_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PACKAGES_IN_BODY_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTITIES_IN_SUBUNIT_REFERENCED | include | reportable | nil | 100 | 0 | 0 | 1 | |
| VARIABLES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| CONSTANTS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 1 | |
| TYPES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| SUBTYPES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENUMERATION_LITERALS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| COMPONENTS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| PROCEDURES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| FUNCTIONS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| IN_PARAMETERS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| OUT_PARAMETERS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| IN_OUT_PARAMETERS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| PACKAGES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| ENTRIES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| EXCEPTIONS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 6 | |
| GENERIC_PROCEDURES_IN_SUBUNIT_RED | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_FUNCTIONS_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |
| GENERIC_PACKAGES_IN_SUBUNIT_REFD | include | reportable | non | 100 | 0 | 0 | 2 | |

# DISTRIBUTION LIST

Codes:

02244
0251
0261
0262 (2)
2094
215
2151 (T. Choinski)
2151 (D. Organ)
2153 (R. Howbrigg)
22
222
2221
2221 (D. Juttelstad) 10 copies
2221 (S. Roodbeen)
2223
2233 (J.McGarry)
38
3891
81
83

Total: 30